

Appendix A

```

(*****
*)
*)      RegistrationUnit
*)
*)      Implements acquisition of data from registration tables, and storage of
*)      same in Touch-Memory
*)
*)      Copyright (c) 1997 Ventana Medical Systems, Inc.  All rights reserved.
*)
*)      Aug 1997 - Martin Lapidus
*)
*)
(*****)

```

unit RegistrationUnit;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
Grids, ExtCtrls, StdCtrls, Buttons,
ErrCodes, // Touch Memory Error codes unit
TMAccess; // Touch Memory Access classes

const

{ set up a user message sent from FormShow }
PART_NUMBER_COLUMN = 0; // For Dispenser string grid on Filling screen
REAGENT_NAME_NUMBER_COLUMN = 1;
MASTER_LOT_NUMBER_COLUMN = 2;
BULK_LOT_NUMBER_COLUMN = 3;
SERIAL_NUMBER_COLUMN = 4;

COMMA = ',';
NO_CUSTOMER_ID = '0'; //Code in database that no Id has been assigned to this kit

WM_AFTERSHOW = WM_USER + 1;
TOUCH_MEMORY_SERIAL_PORT_NUMBER = 1;
MaxDaysForRegistration = 100; // Nexes host can check this and compare to
// Manufacturing date to limit time
// for this package to be registered with Host.
// If value is 0, no limit is imposed.
USER_FILLABLE_LIFE = 18 * 30 + 1; {User fillables (determined by special LOT number)
add 18 months to today's day to create expiration
date. This overrides any date value set for the Master Lot}

type

TTouchMemoryWriteStage = (tmsNone, tmsWriting, tmsVerifying);
TProductType = (ptUnknown, ptKit, ptNonKit_Package);

TRegistrationForm = class(TForm)
Bevel4: TBevel;
Bevel6: TBevel;
FillingStep3Label: TLabel;

```

FillingStep1Label: TLabel;
Label2: TLabel;
Label5: TLabel;
KitBarcodeReaderEdit: TEdit;
StopButton: TBitBtn;
DispensersPanel: TPanel;
Bevel1: TBevel;
Label52: TLabel;
Label56: TLabel;
FillingKitPartNumberLabel: TLabel;
Label64: TLabel;
Label65: TLabel;
FillingKitSerialNumberLabel: TLabel;
FillingKitMasterLotNumberLabel: TLabel;
Label68: TLabel;
FillingDispenserStringGrid: TStringGrid;
FillingFormDoneButton: TBitBtn;
ManualKitDataEntryPanel: TPanel;
Label58: TLabel;
Label57: TLabel;
Label51: TLabel;
FillingSerialNumberEdit: TEdit;
FillingLotNumberEdit: TEdit;
FillingPartNumberEdit: TEdit;
KitDataReadyButton: TButton;
TMStatusLabel: TLabel;
TMLabel: TLabel;
FillingStep1SecondLineLabel: TLabel;
procedure FormShow(Sender: TObject);
procedure FormDestroy(Sender: TObject);
procedure StopButtonClick(Sender: TObject);
procedure KitDataReadyButtonClick(Sender: TObject);
procedure KitBarcodeReaderEditChange(Sender: TObject);
private
    // Touch Memory variables
    sKitExpirationDate      : string;
    sKitPartNumber          : string;
    sKitDescription         : string;
    sMasterLotNumber        : string; // As of 3/11 this is alphanumeric
    sKitSerialNumber        : string; // Alphanumeric
    bWriteToButton          : boolean; // Should this kit be writtent to the Touch Memory?
    nKitSerialNumber        : integer;
    CurrentStep             : integer; // which step out of 3 (only steps 1 and 3 are used here)
    ProductType             : TProductType; // Unknown, Kit, or non-kit package.
    TouchMemory             : TReagentManufacturingTMAccess; // The API
    TouchMemoryRetryCounter : integer; //number of AUTOMATIC retries
    TouchMemoryWriteStage   : TTouchMemoryWriteStage; // (tmsNone, tmsWriting, tmsVerifying);
    // -----Touch memory access -----
    Procedure ShutDownTouchMemoryAccess;
    // Event Handlers for receiving notifications from TReagentManufacutringTMAccess class
    Procedure EHOnTouchMemoryContactMade(TMAccess: TObject);
    Procedure EHOnTouchMemoryContactLostWhileWritingData(TMAccess: TObject);
    Procedure EHOnTouchMemoryWriteOK(TMAccess: TObject);
    Procedure EHOnTouchMemoryWriteError(TMAccess: TObject; ErrorLayer:TErrorLayer;
        ErrorCode : integer; ErrorString : String);

```

```

    Procedure EHTouchMemoryByteWritten(TMAccess: TObject;
        Count:integer;
        TotalBytesToWrite:integer);
    procedure CreateAndConfigureTouchMemoryClass;

    procedure ClearInitialScanEditFields;
    procedure SetupTouchMemoryPanel;
    procedure SetPromptToStep(theStep : integer);
    function ConfirmAndStopActions : boolean;

    { this is a message handler that catches the WM_AFTERSHOW user defined
      message that is posted at the end of FromShow}
    procedure AfterShow(var msg: TMessage); message WM_AFTERSHOW;

    procedure KitDataReady;
    procedure ShowDispensersForThisKit;
    public
    { Public declarations }
    end;

var
    RegistrationForm: TRegistrationForm;

implementation
{$R *.DFM}
uses
    ReagMfgD, {Our Data module}
    ReagMfg2;

    {=====
    SetupTouchMemoryPanel
    1) Collects data for current kit and dispensers.
    2) Creates TouchMemory object.
    3) Writes data to TouchMemory object's buffers
    4) Initiates request to commit data from buffers to TouchMemory device.
    =====}

}

    procedure TRegistrationForm.SetupTouchMemoryPanel;
    var
        NumberOfDispensers : integer;
        isKit                : boolean;
        DataFormatCode      : word;
        ExpirationDate      : string;
        ReagentGroup        : string; // For determining DispenserType
        isPrefilled         : string [1]; // "Y" or "N"
        DispenserType       : string;
    begin
        {is this a kit or a non-kit pack}
        ProductType := ptUnknown;
        with DM.KitMasterConfigQuery do
            begin
                if Locate('KITCONFM_KIT_PART_NUMBER', sKitPartNumber, []) then
                    begin

```

```

sKitDescription := FieldByName('KITCONFM_DESCRIPTION').AsString;
if FieldByName('KITCONFM_IS_PACKAGE_ONLY').AsString = 'Y' then
    ProductType := ptNonKit_Package // One or more dispensers in same package
else
    // that do not form a Kit (ie. master lot
    // lot number is not meaningful.)
    ProductType := ptKit;
end
else
begin
    ShowErrorMsg(Format('Part number %s is not known in the database',[sKitPartNumber]));
    SetPromptToStep(1); // Back to step 1, make initial scan of box
    exit;
end;

end;

isKit := (ProductType = ptKit); // TRUE if kit, otherwise it is a non-kit package
    // (one or more dispenser in package that does
    // not constitute a Kit.)
DataFormatCode := 1; // Code for current data format version as described
    // in the Touch Memory API documentation.
with DM.DispenserCombinedQuery do
begin
    NumberOfDispensers := recordCount;
end;

ExpirationDate := sKitExpirationDate; // All dispensers in kit share kit's expiration date

CreateAndConfigureTouchMemoryClass; // Create the object, assign event handlers,
    // and setup visual cues
// Writing to touch memory is performed in two steps. First data is sent to
// buffers with WriteHeader, WriteKitData, and WriteDispenserData. This is fast.
// Second, data is committed from internal buffers to the Touch Memory with
// RequestCommitDataStringBufferToTouchMemory.
with TouchMemory do //the one just created
begin
    // MaxDaysForRegistration is set as constant, above.
    WriteHeader(DataFormatCode,isKit,NumberOfDispensers,MaxDaysForRegistration);

    // Write kit data, only if needed
    if IsKit then
begin
        WriteKitData(sKitPartNumber,sKitDescription,sKitSerialNumber,
            sMasterLotNumber,sKitExpirationDate);
end;
    // Write Data for each dispenser by stepping through result set for
    //DispenserCombinedQuery. (SQL search for this kit performed above.)

with dm.DispenserCombinedQuery do
begin
    first;
    while not EOF do //all of current SQL result set.
begin
    //Determine DispenserType, used to determine if Antibody fields below

```

```

//are relevant (they are relevant only for Prefilled Anitbody reagents)
ReagentGroup := FieldByName( 'GROUP_NAME' ).AsString;
isPrefilled := FieldByName( 'IS_PREFILLED' ).AsString;
if (UpperCase(ReagentGroup) = 'ANTIBODIES') and (isPrefilled = 'Y') then
  DispenserType := '2'
else
  DispenserType := '1';

WriteDispenserData(DispenserType,
  FieldByName('DISPCONF_REAGENT_NAME').AsString,
  FieldByName('DISPENSE_PART_NUMBER').AsString,
  FieldByName('REAGENT_PRODUCT_CODE').AsString,
  // Name defined in the View for DISPCONF_REAGENT_PRODUCT_CODE

  FieldByName('SERIAL_NUMBER').AsString,
  FieldByName('LOT_NUMBER').AsString,
  FieldByName('MASTER_LOT_NUMBER').AsString,
  FieldByName('IS_PREFILLED').AsString,
  FieldByName('FILLED_DROP_COUNT').AsString,
  FieldByName('LIFE_DROP_COUNT').AsString,
  FieldByName('NOM_DROP_VOLUME').AsString,
  FieldByName('MAX_DROP_VOLUME').AsString,
  FieldByName('DEAD_VOLUME').AsString,
  ExpirationDate, // All dispenser in Kit share the kit's expiration date
  FieldByName('GROUP_NAME').AsString,
  FieldByName('ANTIBODY_CLONE').AsString,
  FieldByName('ANTIBODY_IMM_SUBCLASS').AsString,
  FieldByName('ANTIBODY_PRESENTATION').AsString,
  FieldByName('ANTIBODY_SPECIES').AsString);

  next;
end; // While not EOF
end; // with DM.DispenserCombinedTempQuery
SetPromptToStep(3);
TouchMemoryWriteStage := tmsWriting; // Changed to tmsVerifying later.
RequestCommitDataStringBufferToTouchMemory;
end;
end;

```

```

=====
=
CreateAndConfigureTouchMemoryClass
=====
}

procedure TRegistrationForm.CreateAndConfigureTouchMemoryClass;
begin
  if TouchMemory <> nil then
    begin
      ShowErrorMsg('Touch Memory error. Cannot continue.');// For added safety. Should not occur
      Exit;
    end;

    // Create an instance of the class and assign event handling functions.
    TouchMemory
    TReagentManufacturingTMAccess.Create(TOUCH_MEMORY_SERIAL_PORT_NUMBER);
    :=

```

```

with TouchMemory do
begin
    OnTouchMemoryContactMade := EHTouchMemoryContactMade;
    OnTouchMemoryContactLostWhileWritingData
        := EHTouchMemoryContactLostWhileWritingData;
    OnTouchMemoryWriteOK := EHTouchMemoryWriteOk;
    OnTouchMemoryWriteError := EHTouchMemoryWriteError;
    OnTouchMemoryByteWritten := EHTouchMemoryByteWritten;

    InitializeDataStringBuffer; // Cannot make this a part of WriteHeader in case of need for reentry
    TMLabel.Visible := TRUE;
    TMStatusLabel.Visible := TRUE;
    TMLabel.Caption := '0 )====='; // Represent no-touch state
    TMStatusLabel.Caption := 'Waiting for touch';
    TouchMemoryRetryCounter := 0;
end;
end;

```

```

=====
=
EHTouchMemoryContactMade
Respond to ContactMade event from Touch Memory object by changing prompt.
(Text based graphics shows contact.)

```

```

=====
}
Procedure TRegistrationForm.EHTouchMemoryContactMade(TMAccess: TObject);
begin
    //We may be reentering, so our action depends on where we left off.
    if TouchMemoryWriteStage = tmsWriting then
    begin
        TMLabel.Caption := '0)=====';
        TMStatusLabel.Caption := 'Writing...';
    end
    else
    begin
        TMLabel.Caption := '0)#####'; //Represents writing completed
        TMStatusLabel.Caption := 'Verifying...';
    end;
    Application.ProcessMessages; // Give the captions a chance to appear
end;

```

```

=====
=
EHTouchMemoryContactLostWhileWritingData
Respond to ContactMade lost event from Touch Memory object by
changing prompt and trying again. If number of failed attempts
exceeds preset limit (TIMES_TO_RETRY_CONTACT) then give up and roll back
data previously stored in database.

```

```

=====
}
Procedure TRegistrationForm.EHTouchMemoryContactLostWhileWritingData(TMAccess: TObject);
const
    TIMES_TO_RETRY_CONTACT = 25; // Arbitray. Number of times to automatically try to

```

```

// resume after lost contact.
begin
  TMLabel.Caption := '0 )=====';
  TMStatusLabel.Caption := 'Contact lost. Try again.';

  inc(TouchMemoryRetryCounter);
  // Try again without even notifying the operator
  if TouchMemoryRetryCounter < TIMES_TO_RETRY_CONTACT then
    begin
      TouchMemory.RequestCommitDataStringBufferToTouchMemory;
    end
  else
    begin // Haven't been able to complete the write within TIMES_TO_RETRY_CONTACT
      // automatic retries, so give up and notify the operator.
      // The write may be resumed by starting over again.
      ShowErrorMsg('Unable to maintain contact with Touch Memory. Unable to continue');
      ShutDownTouchMemoryAccess;
    end;
  end;
end;

```

```

=====
ShutDownTouchMemoryAccess
  Free TouchMemory API class and closes the Touch Memory panel. Resets
  prompt to Step 1.
  Called after Successful write to touch memory, or Failed write to Touch Memory,
  or user Abort.
=====

```

```

=}
Procedure TRegistrationForm.ShutDownTouchMemoryAccess;
begin
  TMLabel.Caption := '';
  TMStatusLabel.Caption := '';
  TouchMemory.Free;
  TouchMemory := nil;
  SetPromptToStep(1); // Back to step 1, make initial scan of box
end;

```

```

=====
EHOnTouchMemoryWriteOK
  Event handler signaling success in writing, verifying, and write protecting
  the Touch Memory.
  After we are done writing to the touch memory, we mark the KIT_WAS_REGISTERED
  field in our database.
=====

```

```

=}
Procedure TRegistrationForm.EHOnTouchMemoryWriteOK(TMAccess: TObject);
begin
  ShowInfoMsg('Data successfully written to Touch Memory');
  TouchMemoryWriteStage := tmsNone; // neither writing nor verifying.
  ShutDownTouchMemoryAccess; // Cleans up and Frees the class
  with DM.KitsTable do
    begin

```

```

if
Locate('KITS_KIT_PART_NUMBER;KITS_MASTER_LOT_NUMBER;KITS_KIT_SERIAL_NUMBER',
      VarArrayOf([sKitPartNumber,sMasterLotNumber,sKitSerialNumber]), []) then
    begin
        // Safety only. If we should not even be here if this record could not be found.
        ShowErrorMsg('Unable to record Touch Memory burn to database.');
```

```

    end
else
    begin
        Edit;
        FieldByName('KITS_WAS_REGISTERED').AsString := 'Y';
        Post;
    end;
end;
end;

{=====
=====
EHTouchMemoryByteWritten
Event handler signaling another byte successfully written to touch memory.
We only update display every 10 bytes.
We currently use a text based graphical display.
=====
=====
=}

Procedure TRegistrationForm.EHTouchMemoryByteWritten(TMAccess: TObject;
      Count:integer; // number of bytes written
      TotalBytesToWrite:integer);

var
    TentsCompleted : integer;
    s                : string;
begin
    if count = TotalBytesToWrite then // Are we done with writing? (Verification next)
    begin
        TLabel.Caption := '0)#####'; // Done Writing
        TMSStatusLabel.Caption := 'Verifying data';
        TouchMemoryWriteStage := tmsVerifying;
    end
else
    begin
        // Update display every 10 bytes
        if (Count mod 10 = 0) then
        begin
            TentsCompleted := round(Count / TotalBytesToWrite * 10);
            s := copy('#####',TentsCompleted,10); // Take 10 symbols from
                // a starting point that
                // moves each tenth.
            TLabel.Caption := '0)' + s; // Will typically look like 0)#####
        end;
    end;
    Application.ProcessMessages; // Get the label a chance
end;

{=====
=====
EHTouchMemoryWriteError

```


Event handler signaling a non-recoverable error while writing to or verifying the Touch Memory.

=}

```

Procedure TRegistrationForm.EHOnTouchMemoryWriteError(TMAccess : TObject;
               ErrorLayer : TErrorLayer;
               ErrorCode : integer;
               ErrorString : String);

begin
  ShowErrorMsg('Touch Memory error #' + IntToStr(Ord(errorLayer))
    + '/' + intToStr(ErrorCode) + ': ' + ErrorString + '. Unable to continue.');
```

```

  ShutDownTouchMemoryAccess; // Cleans up and Frees the class
end;
```

==

```

SetPromptToStep
Moving to a specific step (1 of 3). Show the appropriate label and
enable to appropriate buttons.
```

=====}

```

procedure TRegistrationForm.SetPromptToStep(theStep : integer);
begin
  CurrentStep      := theStep;
  FillingStep1Label.Enabled := (theStep = 1);
  FillingStep1SecondLineLabel.Enabled := (theStep = 1);
  {There is NO fillingStep2 on this form.}
  FillingStep3Label.Enabled := (theStep = 3); {This label may be hidden according
    to the KITCONFM_WRITE_TO_BUTTON
    field for the scanned kit.
    (ES kits do not have buttons)
  }

  // Only allow operator to exit if we are at step 1. Otherwise operator must
  // first click Stop to confirm and then return to step 1.
  FillingFormDoneButton.Enabled := (theStep = 1);
  StopButton.Enabled := Not FillingFormDoneButton.Enabled;
  if theStep = 1 then
    begin
      FillingStep3Label.visible := TRUE; // Visible unless we later find the kit does not get written
      KitBarCodeReaderEdit.Text := ''; // Prepare for next time around
      KitBarCodeReaderEdit.SetFocus; // Make sure bar code scanner output
        // goes here
      DispensersPanel.Hide;

    end;
  end;
```

```

procedure TRegistrationForm.StopButtonClick(Sender: TObject);
begin
  ConfirmAndStopActions; // Ignore returned result
end;
```

```

=====
ConfirmAndStopActions
  If we have not yet comitted any data to the database, then just ask for
  confirmation and close panels.
  If we have comitted data, ask for confirmation, roll back from database, close
  panels.

  return TRUE if user confirms the Stop.

  Called in response to Stop button click and FormCloseQuery
=====
}

```

```

function TRegistrationForm.ConfirmAndStopActions : boolean;
begin
  result := FALSE; // The Stop has not yet been confirmed and completed.

  if CurrentStep = 1 then //No operation is pending, so we can close without confirmation
  begin
    result := TRUE;
    exit;
  end;
  {Not step 1, so we must be talking to the touch memory}
  if MessageDlg('Do you really wish to cancel writing to the Touch Memory?',
    mtConfirmation,[ mbYes, mbNo ],0 ) = mrYes then
  begin
    TouchMemory.AbortAction; // Kills a timer
    // Continue with other Stop steps, below
  end
  else
    exit; // User wants to continue

  DispensersPanel.Hide;
  ShutDownTouchMemoryAccess; // Back to step 1.
  result := TRUE;
end;

```

```

=====
AfterShow
  Respond to WM_AFTERSHOW message posted by FormShow. The form is now visible
=====
}

```

```

procedure TRegistrationForm.AfterShow(var msg: TMessage); {message WM_AFTERSHOW;}
begin
  {No harm in trying to load the DLL if it has already been loaded. That is
  handeled by the function}
  if NOT TMLoadTouchMemoryDLL then
  begin
    ShowMessage('Cannot load required DLL: IBFS32.DLL');
    close;
    exit;
  end;
end;

procedure TRegistrationForm.ClearInitialScanEditFields;

```

```

begin
  FillingLotNumberEdit.Text := "";
  FillingSerialNumberEdit.Text := "";
  FillingPartNumberEdit.Text := "";
end;

```

```

=====

```

```

FormShow
  Called when form is displayed. Performs some initialization.

```

```

=====
}

procedure TRegistrationForm.FormShow(Sender: TObject);
begin
  {only show manual entry fields if operator can reburn a button}
  ManualKitDataEntryPanel.visible := faReburnTouchMemory in AccessableFeatures;
  ClearInitialScanEditFields;
  SetPromptToStep(1); // Instruct operator to Scan
  // Make sure secondary panels are not visble.
  DispensersPanel.Hide;
  {Want to check for presence of DLL after FormShow has been completed so we
  can shut down if necessary.}
  PostMessage(Handle, WM_AFTERSHOW, 0, 0);
end;

```

```

=====

```

```

FormDestroy
  Called when form is destroyed. Frees the TouchMemory object, if necessary.

```

```

=====
}

procedure TRegistrationForm.FormDestroy(Sender: TObject);
begin
  TouchMemory.Free; //Destroy the TouchMemory object if it was created.
  // No harm if it was not.
  TouchMemory := nil;
end;

```

```

=====

```

```

KitDataReady
  Product code, master lot, and serial number have been read from bar code by
  scanner. Extract these from edit fields and test for validity.
  (Note: edit fields are hidden except for development and testing.)

  Identify Kit and display the dispensers it contains

```

```

=====
}

procedure TRegistrationForm.KitDataReady;
var
  bPreviouslyRegistered : boolean;
  tempExpirationDate   : TDateTime;

```

```

begin
if not GetAndValidatePartNumberFromEdit(FillingPartNumberEdit,
                                         sKitPartNumber) then
begin
    KitBarcodeReaderEdit.SetFocus;// GetAndValidatePartNumberFromEdit may set
                                // the focus to FillingPartNumberEdit if an
                                // invalid value is found. Make
                                // sure bar code scanner output goes to
                                // KitBarcodeReaderEdit.

    exit;
end;

sMasterLotNumber := FillingLotNumberEdit.Text;
sKitSerialNumber := FillingSerialNumberEdit.Text;
ClearInitialScanEditFields;
if NOT isValidLotNumber(sMasterLotNumber) then
    exit; // Message will have been shown.

nKitSerialNumber := ConvertToPositiveInteger_WithMessageIfFail(sKitSerialNumber);
if nKitSerialNumber <= 0 then
    exit;

ProductType := ptUnknown;
// Need additional information about the kit:
// 1) Description (name)
// 2) Is this the part number of a kit?
with DM.KitsTable do
begin
    if
        NOT
        Locate(
'KITS_KIT_PART_NUMBER;KITS_MASTER_LOT_NUMBER;KITS_KIT_SERIAL_NUMBER',
    varArrayOf([sKitPartNumber,sMasterLotNumber,sKitSerialNumber]), []) then
begin
    ShowErrorMsg(Format('Kit Part # %s, Lot # %s, Serial # %s is not known in the database',
        [sKitPartNumber,sMasterLotNumber,sKitSerialNumber]));
    SetPromptToStep(1); // Back to step 1, make initial scan of box
    exit;
end
else
begin
    bPreviouslyRegistered := FieldByName('KITS_WAS_REGISTERED').AsBoolean;
    if bPreviouslyRegistered then {Operator MAY have rights to Reburn button}
begin
    if faReburnTouchMemory in AccessableFeatures then { as set at Log in }
begin
        if MessageDlg('This kit has previously been registered. You are authorized to write a second
button for this kit.'
            +' Do you want to write a second button for this kit?',
            mtInformation, [mbYes, mbNo], 0) = mrNo then
begin {does not want to reburn}
                SetPromptToStep(1); // Back to step 1, make initial scan of box
                exit;
            end; {otherwise continue with 2nd burn of this button}
        end
    else {Operator not authorized to reburn button}
begin

```

```

        ShowErrorMsg('This kit has previously been registered.');
```

only

```

        SetPromptToStep(1); // Back to step 1, make initial scan of box
        exit;
    end;
end;
// To get expiration date, check Master lots table keying on
// Master Lot Number (several kits may share this, so locate may
// not return a unique record, but all will have the same lot number.
with DM.MasterLotsMDisplayTable do
begin
    if NOT Locate( 'MASTLOTM_MASTER_LOT_NUMBER', sMasterLotNumber, [] ) then
    begin
        ShowErrorMsg('Cannot find expiration data for ' + sMasterLotNumber + '.');
        SetPromptToStep(1); // Back to step 1, make initial scan of box
        exit;
    end
    else
    begin
        sKitExpirationDate := FieldByName('MASTLOTM_EXPIRATION_DATE').AsString;
        {NOTE: The short date format is initially set in ReagMfg1.pas at startup.
        We test it here only to catch unexpected (and not-understood) changes. 7/22/97}
        if ShortDateFormat <> 'mm/dd/yyyy' then
        begin
            ShowMessage('Current date format is : '+ShortDateFormat+ '. Changing to mm/dd/yyyy.');
```

only

```

            ShortDateFormat := 'mm/dd/yyyy';
        end;
        sKitExpirationDate := DateTimeStringToDateOnlyString(sKitExpirationDate);//make it date

        if Uppercase(sMasterLotNumber) = 'USER' then
        begin
            tempExpirationDate := now + USER_FILLABLE_LIFE;
            sKitExpirationDate := dateToStr(tempExpirationDate);
        end;
        // Set up and show panel for Kits
        ShowDispensersForThisKit;
    end;
end;
end;
end;
end;
end;
```

```

procedure TRegistrationForm.KitDataReadyButtonClick(Sender: TObject);
begin
    KitDataReady;
end;
```

```

procedure TRegistrationForm.ShowDispensersForThisKit;
var
    GridRow      : integer;
    sDispenserPartNumber : string;
begin
    DispensersPanel.Hide; // Will be made visible if scanned data is validated
```

```

// Fill Labels with data acquired from Kit bar code
FillingKitPartNumberLabel.caption := sKitPartNumber;
FillingKitSerialNumberLabel.caption := sKitSerialNumber;
FillingKitMasterLotNumberLabel.caption := sMasterLotNumber;
with FillingDispenserStringGrid do
begin
    // Set up column headings and widths(?) for dispenser grid
    cells[PART_NUMBER_COLUMN,0] := 'Part Number';
    cells[SERIAL_NUMBER_COLUMN,0] := 'Serial Number';
    cells[MASTER_LOT_NUMBER_COLUMN,0] := 'Master Lot';
    cells[BULK_LOT_NUMBER_COLUMN,0] := 'Bulk Lot';
    cells[REAGENT_NAME_NUMBER_COLUMN,0] := 'Reagent';

    ColWidths[PART_NUMBER_COLUMN] := 65;
    ColWidths[MASTER_LOT_NUMBER_COLUMN] := 65;
    ColWidths[SERIAL_NUMBER_COLUMN] := 75;
    ColWidths[BULK_LOT_NUMBER_COLUMN] := 65;
    // Give all the rest of the space to the reagent name column
    ColWidths[REAGENT_NAME_NUMBER_COLUMN] := ClientWidth
        - ColWidths[PART_NUMBER_COLUMN]
        - ColWidths[SERIAL_NUMBER_COLUMN]
        - ColWidths[MASTER_LOT_NUMBER_COLUMN]
        - ColWidths[BULK_LOT_NUMBER_COLUMN];

end;

// Select only those Dispensers associated with this Kit Part Number and Master Lot number
// and Serial Number

with DM.DispenserCombinedQuery do
begin
    Close;
    with SQL do
        begin
            Clear;
            Add('SELECT *');
            Add('FROM ":' + SystemAlias + ':' + DispenserDataViewName + '"');
            Add('WHERE ');
            Add('DISPENSE_KIT_SERIAL_NUMBER = ' + sKitSerialNumber );
            Add('AND ');
            Add('KIT_PART_NUMBER = ' + sKitPartNumber + '"');
            Add('AND ');
            Add('MASTER_LOT_NUMBER = ' + sMasterLotNumber + '"');
        end;
    Open;
    if RecordCount = 0 then
        begin
            ShowErrorMsg(Format(
                'Cannot find Dispensers for Kit Part # %s, and Master Lot # %s, and Serial # %s',
                [sKitPartNumber,sMasterLotNumber,sKitSerialNumber]));
            exit; // Nothing to do so leave
        end;
    // Fill grid with available data from table. Incomplete fields will be supplied
    // from individual dispenser bar codes.
    with FillingDispenserStringGrid do

```

```

begin
  RowCount := RecordCount + 1; // Row 0 of grid is used for captions, not record data
  // Use DispenserCombinedQuery to acquire data
  First; // First record in MasterLotsQuery
  GridRow := 1;
  While Not EOF do // until last record of query processed
    begin
      sDispenserPartNumber := FieldByName('DISPENSE_PART_NUMBER').AsString;
      Cells[PART_NUMBER_COLUMN,GridRow] := sDispenserPartNumber;
      cells[REAGENT_NAME_NUMBER_COLUMN,GridRow] :=
FieldByName('DISPCONF_REAGENT_NAME').AsString;
      Cells[MASTER_LOT_NUMBER_COLUMN,GridRow] :=
FieldByName('MASTER_LOT_NUMBER').AsString;
      Cells[BULK_LOT_NUMBER_COLUMN,GridRow] :=
FieldByName('LOT_NUMBER').AsString;
      Cells[SERIAL_NUMBER_COLUMN,GridRow] :=
FieldByName('SERIAL_NUMBER').AsString;
      Next; // Next record of query
      inc(GridRow);
    end;
    Height := (RowCount) * (DefaultRowHeight + 1)+ 3;//Adjust height of grid on form
    if Height > DispensersPanel.Height - top - 10 then // Don't let the grid
      Height := DispensersPanel.Height - top - 10 // extend below panel
    end; //With Grid
  end; // With Query
  // If we are here, data acquired from bar code reader is valid w.r.t. our database
  // so we can show the panel and continue.
  DispensersPanel.Show;

  KitBarCodeReaderEdit.Text := ""; // Prepare for next time around
  KitBarCodeReaderEdit.SetFocus; // Make sure bar code scanner output
    // goes here.
  ClearInitialScanEditFields;
  SetupTouchMemoryPanel;
end;

=====
}

KitBarCodeReaderEditChange
Respond to change in KitBarCodeReaderEdit.
Verifies that bar code reader has made the change by looking for preprogrammed
prefix character ('!').
If text closes with suffix character ('!') then entry is complete. It is parsed
into 3 strings: Part number, Master Lot/Lot number, and serial number. These
3 substrings are then copied to edit fields as if they were individually entered
manually.

=====
}

procedure TRegistrationForm.KitBarCodeReaderEditChange(Sender: TObject);
const
  PREFIX_CODE = '!';
  SUFFIX_CODE = '!';

  PART_NUMBER_FIELD_LENGTH = 8;
  LOT_NUMBER_FIELD_LENGTH = 8;

```

```

SERIAL_NUMBER_FIELD_LENGTH = 4;

FULL_LENGTH = PART_NUMBER_FIELD_LENGTH + LOT_NUMBER_FIELD_LENGTH
              + SERIAL_NUMBER_FIELD_LENGTH;

var
FullString : string; // As obtained from scanner
len        : integer; // Length of string obtained.
S1,S2,S3   : string; // Parsed substrings
begin
fullString := KitBarcodeReaderEdit.text;
len        := length(fullString);
// If somehow entry is coming from keyboard, prefix will be wrong; clear
// the entire input string.
if (len>0) and (FullString[1] <> PREFIX_CODE) then
begin
KitBarcodeReaderEdit.Text := ""; // Throw away the incorrect input
exit;
end;

if length(FullString) < FULL_LENGTH + 2 then //Additional 2 for Prefix and Suffix codes
begin
// If the string ends with SUFFIX_CODE, but is not full length, then we
// probably scanned a dispenser label (it is shorter)
if (len > 1) and (fullString[len] = SUFFIX_CODE) then
begin
KitBarcodeReaderEdit.Text := ""; // Throw it all away
end;
exit; //Just leave. Except for the special case, will keep incomplete string in tact.
end;

// Check last character to see if is final delimiter. If not just exit to
// wait for additional characters.
if FullString[length(FullString)] <> SUFFIX_CODE then
begin
KitBarcodeReaderEdit.Text := ""; // Throw away the incorrect input
exit;
end;

// If we are here we have the correct prefix and suffix codes.

// Make sure the length of string is what we expect. If not, then warn and exit
if length(FullString) <> FULL_LENGTH + 2 then // Extra 2 for Prefix and Suffix
begin
ShowErrorMsg(FullString + ' is not in the correct format. ');
KitBarcodeReaderEdit.Text := ""; // Throw away the incorrect input
exit;
end;
//Strip off Prefix and Suffix characters, then parse according to sub field lengths
FullString := Copy(FullString,2,FULL_LENGTH);
S1 := copy(FullString,1,PART_NUMBER_FIELD_LENGTH);
S2 := copy(FullString,1 + PART_NUMBER_FIELD_LENGTH,LOT_NUMBER_FIELD_LENGTH);
S3 := copy(FullString,1 + PART_NUMBER_FIELD_LENGTH + LOT_NUMBER_FIELD_LENGTH,
          SERIAL_NUMBER_FIELD_LENGTH);
FillingPartNumberEdit.Text := trim(S1);

```



```

FillingLotNumberEdit.Text := trim(S2);
FillingSerialNumberEdit.Text := trim(S3);
KitBarcodeReaderEdit.Text := "; // Prepare for next time around. Will cause
                                // Reentry here
KitDataReady;
end;

end.

```

TO: 646666